information

Article

A Guided Scratch Visual Execution Environment to Introduce Programming Concepts to CS1 Students

Raquel Hijón-Neira [1], Cornelia Connolly [2,3,*], Daniel Palacios-Alonso [1] and Oriol Borrás-Gené [1]

[1] Department of Computer Science, Universidad Rey Juan Carlos, 28032 Madrid, Spain; raquel.hijon@urjc.es (R.H.-N.); daniel.palacios@urjc.es (D.P.-A.); oriol.borras@urjc.es (O.B.-G.)
[2] School of Education, National University of Ireland Galway, H91 TK33 Galway, Ireland
[3] Lero, SFI Centre for Software Research, National University of Ireland Galway, H91 TK33 Galway, Ireland
* Correspondence: cornelia.connolly@nuigalway.ie

Abstract: First-year computer science (CS1) university students traditionally have difficulties understanding how to program. This paper describes research introducing CS1 students to programming concepts using a Scratch programming language guided visual execution environment (VEE). The concepts addressed are those from an introductory programming course (sequences, variables, operators, conditionals, loops, and events and parallelism). The VEE guides novice students through programming concepts, explaining and guiding interactive exercises executed in Scratch by using metaphors and serious games. The objective of this study is, firstly, to investigate if a cohort of 124 CS1 students, from three distinct groups, studying at the same university, are able to improve their programming skills guided by the VEE. Secondly, is the improvement different for various programming concepts? All the CS1 students were taught the module by the same tutor in four 2-h sessions (8 h), and a qualitative research approach was adopted. The results show students significantly improved their programming knowledge, and this improvement is significant for all the programming concepts, although greater for certain concepts such as operators, conditionals, and loops than others. It also shows that students lacked initial knowledge of events and parallelism, though most had used Scratch during their high school years. The sequence concept was the most popular concept known to them. A collateral finding in this study is how the students' previous knowledge and learning gaps affected grades they required to access and begin study at the university level.

Keywords: programming; visual execution environment; CS1; metaphors

Citation: Hijón-Neira, R.; Connolly, C.; Palacios-Alonso, D.; Borrás-Gené, O. A Guided Scratch Visual Execution Environment to Introduce Programming Concepts to CS1 Students. *Information* **2021**, *12*, 378. https://doi.org/10.3390/info12090378

Academic Editor: Enrico Denti

Received: 27 August 2021
Accepted: 13 September 2021
Published: 17 September 2021

1. Introduction

Computer programming education is key to the acquisition of 21st-century skills such as creativity, critical thinking, problem solving, communication and collaboration, social–intercultural skills, productivity, leadership, and responsibility [1]. Studies in many countries report using Scratch or games [2–4]. That being so, it is still unclear the best order in which to introduce programming concepts to CS1 students. There are difficulties teaching basic concepts such as program construction [5], loops [6], control structures, and algorithms [7]. Difficulties may arise from poor or even a lack of a proper teaching methodology [8,9], and teachers need some guidance to approach this task efficiently [10,11].

Papert argued that a child able to program a computer would be able to gain an actionable understanding of probabilistic behavior, as, through such activity, they would be connected with empowering knowledge about the way things work [12]. His view was that programming was a way to connect the programmer with cognitive science, in that programming enables one to articulate ideas explicitly and formally and to see whether the idea works or not. Papert encouraged one to 'look at programming as a source of descriptive devices' [12], predicting that, 'in a computer-rich world, computer languages that simultaneously provide a means of control over the computer and offer

new and powerful descriptive languages for thinking will . . . have a particular effect on our language for describing ourselves and our learning' (p. 98). Affected by the learning procedure, instructional materials, and technology used, along with metacognitive factors, there are many variables involved in learning to code. Others emphasized that well-designed lessons with interesting activities become meaningful only when they affect the students in the process [13]. It was also asserted that innovations in the methods of teaching and the use of teaching aids may improve students' feeling of success [14] and may help them develop confidence, which correlates with the practices and theories of Piaget and Vygotsky and the adoption of constructivism in teaching [15–19].

Many approaches have been implemented to help students learn programming for the first time, for example with different devices such as using mobile devices [20] or different methodologies such as pair programming [21]. Students have traditionally encountered difficulties and misconceptions on the concepts learned [22]. Along with the learning of programming, researchers and educators in higher education aim to improve student computational thinking (CT) skills using appropriate interventions [23] and approaches using games for teaching and learning, CT principles, and concepts [24].

In response to this proposition, the contribution of this paper is a rigorous study aimed at determining a satisfactory way to introduce basic programming concepts and CT at the CS1 level and inquire how this affects students' learning gains. This paper evaluates a Guided Scratch Visual Executing Environment developed for CS1 students as a method to teach, develop, practice, and learn computer programming. To address this, there are two research questions.

1. Can programming concepts be improved with a Visual Execution Environment for this cohort for CS1 students?
2. Which of the programming concepts tend to be more easily understood, and which are more difficult?

The concepts addressed here are those in a traditional 'Introduction to Programming' course such as: sequences, flowcharts, variables and types of data, operators, conditionals, loops, events, and parallelism.

This research proposal investigates if a cohort of 124 CS1 students, from three distinct groups, studying at the same university, are able to improve their programming skills guided by the VEE. Secondly, it investigates if the improvement varies for different programming concepts. The CS1 students were taught the module by the same tutor, and the procedure included a review of material in an interactive way using the Guided Scratch VEE, where there was an explanation of the concepts with prepared ad-hoc exercises based on metaphors for each concept and practice with proposed exercises in Scratch. This was conducted in four 2-h sessions (8 h). There was a pre- and post-test evaluation to measure the gains in students' learning. The same test was used for the pre-test and the post-test, consisting of 27 short-answer questions covering the programming concepts addressed. The results demonstrate students significantly improved their programming knowledge, and this improvement is significant for all the programming concepts, although greater for certain concepts such as operators, conditionals, and loops. Results also demonstrate that students lacked initial knowledge of events and parallelism, although most had used Scratch in high school. The sequence concept was the most popular. A collateral finding with the study is how the students' previous knowledge and learning gap affected the grades with which they gain access to study at university.

2. Theoretical Framework

2.1. Learning Programming

It has been argued that programming is simply a very difficult subject or skill, and therefore, it is unsurprising that students find it challenging [25]. Others have investigated the factors that indicate students' ability to learn programming, demonstrating mathematical ability, processing capacity, analogical reasoning, conditional reasoning, procedural

thinking, and temporal reasoning [26]. Therefore, a competent programmer must master multiple skills.

Brooks [27] states, "I believe the hard part of building software to be the specification, design and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared to the conceptual errors in most systems. If this is true, building software will always be hard." (p. 182). First-year computing students (CS1), in their first semester of learning how to program, are likely to encounter many of the challenges Brooks highlights. Apart from understanding the algorithm design process and algorithm creation, novice programmers are required to master abstract conceptualization, the programming language, and the environment in which they will be working.

Many CS1 students are introduced to the simplest of computer programs, where only the basic concepts of the main function and system output are presented, such as 'hello world'. Additional components are included over the course of the term, and by the end, students have gradually been introduced to the complete language. Students are expected to practice the use of the programming concepts by undertaking various exercises, until they have demonstrated an ability to author programs given specifications [28].

The process of transforming system design specifications into working program code is sequential, involving five particular components: specification, algorithm, design, code, and test. The specification (normally written in plain language) draws the students to an understanding of the problem domain and devises an appropriate algorithm, often ensuring the specification is re-written in a precise manner that is close to implementation. Drawing heavily on abstraction, the algorithm is translated into programming concepts through design and subsequently into actual code. Given a correct design, this stage should not be challenging and is determined by the programming language. The final stage is that of testing, which leads to the implementation of the program. This sequence is the most appropriate in developing an efficient computer program; however, students often leap into the final 'code' section before any specification or design takes place. Many novice programmers tend to concentrate on syntax [29], as, often, such approaches are reinforced in the manner the topic is presented in both lectures and books.

Programming draws on many skills such as problem solving, abstraction, mathematics logic, procedural, testing, debugging, and professional development [26], and these skills cannot be applied in isolation. They are applied in the context of a particular problem or problem area. In an educational environment, the degree program a student is following will often determine the amount of programming conducted, the language, and the environment.

2.2. Complementary Approaches for Teaching Programming and CT

Many approaches have been identified to help students trying to master computer programming when they are first introduced to it, such as by using an adaptive virtual reality platform [30], as well as through solving problems in artificial intelligence [31], simulation games [32], serious games [33,34], using robots [35], or comparisons between block or text programming [36,37]. In this vane, other important contributions have been highlighted for helping to teach CT to students using robotics [38] or adaptive scaffolding for learner modeling [39]. Another approach includes hands-on projects using R with application to mathematics [40], such as the Lab Rotation Approach [41]. On the other hand, methods that try to help teachers improve their CT [42,43] include storytelling [44] or, as previously, using robotics [45,46].

2.3. The Visual Execution Environment

The Scratch Visual Execution Environment (VEE) proposed in this paper utilizes pre-established Scratch programs, which include the theory and practice corresponding to each of the proposed lessons. It is a web application that can be accessed from any device apart from a PC (smartphones, tablets, etc.). Such block-based coding environments

prevent syntax errors while ensuring concepts and practices foundational to programming are uncovered. These environments are comfortable for novices at programming and help develop their computational problem-solving skills, which can be used to generate a wide spectrum of problems from easy to difficult.

Structuralist theory considers that a game establishes the way of seeing the world and thinking of the participant. The absence of this 'learn to think' prevents further learning from having depth (they are not reflexive), and therefore, it does not activate the emotional part that enables long-term learning [47]. On the other hand, the Fogg model [48], designed to change human behavior, establishes that three elements are necessary to modify human behavior: the motivation (individuals feel inclined to perform an action for pleasure, fear, recognition, rewards, etc.); the skill (the level of difficulty perceived by the person to perform the action), and the trigger (the agent that triggers the behavior). This game facilitates a dynamic in which these three components converge simultaneously, an optimal method for teaching–learning dynamics of new concepts, which, in our case, comprises concepts related to programming for the CS1 cohort.

The VEE draws on the TPACK model by Mishra and Koehler [49] in integrating the necessary knowledge and the development of a useful tool to transmit the programming concepts [50]. TPACK defines the area in which technology is consistently integrated into teaching, and the transfer of knowledge to the student is enhanced. This area is the intersection of three fields of knowledge: Content Knowledge (programming concepts), Pedagogical Knowledge (demonstration and serious game integration), and, thirdly, Technological Knowledge (coding with Scratch and the development of web pages). At the intersection of the three domains is TPACK, as demonstrated in Figure 1.

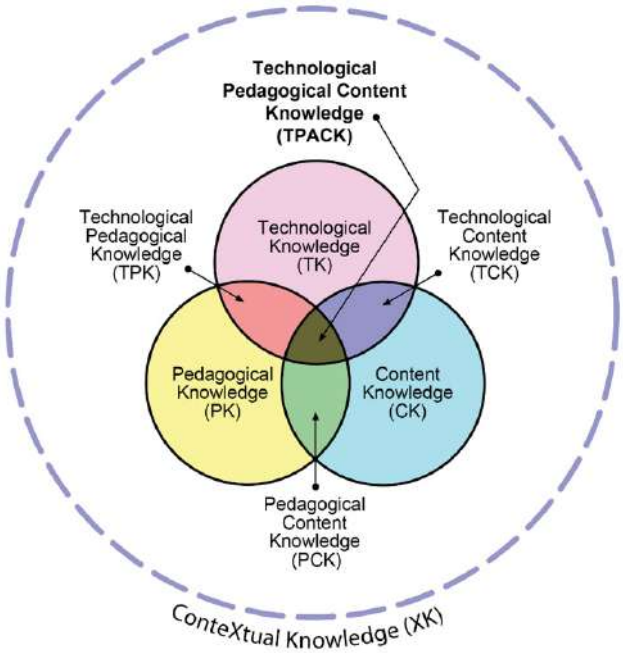

Figure 1. TPACK model [51].

3. Research Method

The didactic research approach in computing education, adopted in this study, was developed based on the principles of research-based learning to cultivate the students' skills in developing CS skills. The quasi-experimental procedure with pre- and post-test was followed. For the pre- and post-tests, the same evaluation tests were used for programming and computational thinking. Each test was completed individually by each student in their 'Introduction to Programming' class.

3.1. Pedagogical Approach

The seven computational concepts proposed by Brennan and Resnick [50], which were developed in differentiated applications, allow coherent sequencing without mixing concepts. An eighth theme was included: the concept of computational thinking detailed. The TPACK Scratch Visual Execution Environment has pre-established programs, which include the theory and practice corresponding to each of the proposed lessons. This separation allows the teacher to use different sequencing from the one proposed, if necessary. Since some concepts are supported by prior learning, it is necessary, e.g., to explain the operation of conditionals, it is necessary to previously understand logical operators. The order of topics proposed is presented in Table 1.

Table 1. Proposal for sequencing topics in the Guided Scratch VEE.

Lesson Number	Topic
Lesson 1.	Sequences
Lesson 2.	Variables
Lesson 3.	Operators
Lesson 4.	Conditionals
Lesson 5.	Loops
Lesson 6.	Events
Lesson 7.	Parallelism
Lesson 8.	Computational Thinking

Use of Visual Metaphors

The use of metaphors for educational purposes dates back to ancient times (e.g., Plato's Dialogues), where a known concept is transferred from one object to another, to provide a new notion or intuition [52]. In this work, attempts have been made to make metaphors evident, and, where possible, graphic representations have been used to facilitate assimilation. Below is a list of the main metaphors used (Table 2).

Table 2. Metaphors of programming concepts in the Guided Scratch VEE.

Concept	Metaphor
Sequence	Cooking recipe
Variable	Container with label
Conditional	Detour on the road
Loop	How a clock works
Event	Traffic light operation
Synchronization	Set the same time on two watches
Computational thinking	NIM game

In the case of the 'Operator' concept, it has not been considered necessary to evoke a metaphor, since the notion of a mathematical operator is widely extended. The metaphor associated with computational thinking is the NIM game, a game that had its origin in China, although its first reference was in Europe in the sixteenth century. The current name was coined by Bouton of Harvard University, who developed the full game theory—a mathematical game in which the player who starts the game always wins if they know the rule to find the solution of each movement: exclusive disjunction by digits in binary [53].

The topics developed are closely related to the computational concepts implicit in Scratch as a programming initiation language [28]. Therefore, the first seven themes are valid tools for learning programming. The last topic includes the notion of computational thinking as an exercise of recapitulation and reinforcement of the previous points. Computational thinking is a complex competence that is related to the mental schemes of human

beings, which allows the development of ideas and links abstraction (ideas–concepts) with pragmatism (action). It is not synonymous with programming, since it requires different degrees of abstraction and does not depend on computer equipment (unplugged). However, the use of computer equipment allows us to undertake tasks that without them would be unapproachable [54].

In the development of each topic, it was considered that the best way for the assimilation of the concept to be treated was to first provide an exhibition and then carry out several exercises or practice sessions on the concept. To highlight this separation, a visual key was used as a resource. The exhibition part has a classic slate background, and the practical part has a grid notebook background (Figure 2). In the practical element, the combined interaction with the treated concept permits instant feedback, allowing both the assimilation and the accommodation of the new concepts (Figure 3).

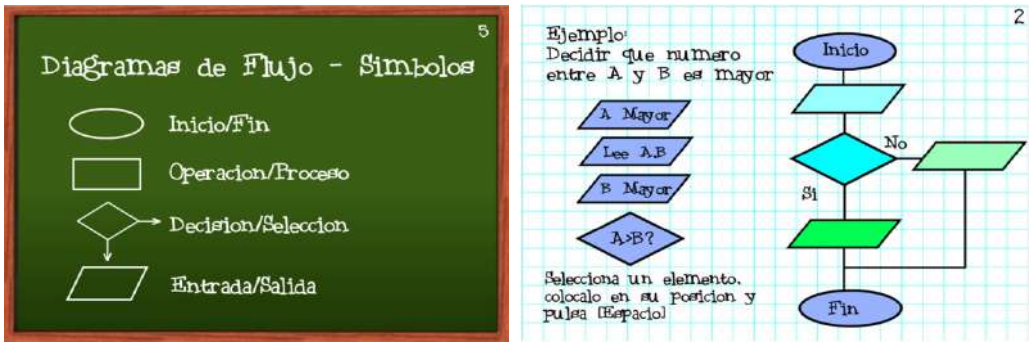

Figure 2. Examples of exposure (**left**) and practice (**right**) in the TPACK Guided Scratch VEE.

Figure 3. The TPACK Guided Scratch VEE with the programming concepts and some of their Scratch programs. Last option is Local Scratch.

3.2. Research Participants

The research participants are three cohorts of CS1 university students, totaling 124, studying the same Video Games Design and Development undergraduate degree program divided into two campuses in the Madrid city catchment area. All were enrolled in an 'Introduction to Programming' module in the first semester of their degree. The research was carried out at a public university in Madrid with two differentiated campuses: one located in the city center and the other in a nearby town.

The breakdown of the 124 university students was 28 students (22.6%) from the double degree with computers (G1) from the Mostoles campus (nearby town to Madrid city), 41 students (33.1%) from the single degree in the Madrid campus (G2), and 55 students (44.4%) from the single degree in the Móstoles campus (G3). Student ages ranged from 17 to 18 years. The CS1 students were taught the module by the same tutor who had a computer science Ph.D. and more than 18 years' experience in teaching this particular programming module. The classroom assistants also had a computer science Ph.D. and teaching experience, and they assisted all three groups.

The 'Introduction to Programming' module was taught in the same manner as the three classes, avoiding object-oriented programming, as that was not the purpose of this course. The work in class during the first two weeks of the course was a brief introduction to programming, a text-oriented programming language, where the participants were introduced to the basic concepts of sequences, variables, and expressions. In the third week of teaching, visual programming with Scratch was added for a short period of time, as required by the degree syllabus. The procedure included a review of material in an interactive way using the Guided Scratch VEE, where there was an explanation of the concepts with prepared ad-hoc exercises based on metaphors for each concept and practice with proposed exercises on Scratch. The research study was conducted in four 2-h sessions (8 h). The evaluation used for the pre-test and post-test was the same: 27 short-answer questions covering the programming concepts addressed in Table 1. The scoring rubric graded each question differently based on the complexity and completeness of the concepts, ranging from either from 0 to 1 or from 0 to 2.

4. Findings and Discussion

In presenting the outcomes of the research, we will first present results from the cohort in its entirety, the 124 students in total. The second approach was a subdivision of the first, subdivided into the two campus locations, Ferraz and Mostoles, and degree type, just one degree (Design and Development of Video Games at Ferraz and Mostoles) or two at the one time (dual Design and Development of Video Games and a degree in Computer Science). Each cluster was based on the degree level and the campus location of the student. Students were enrolled in the degree of Design and Development of Video Games or on a dual degree studying Design and Development of Video Games and Computer Science.

Students belonged to the campus in the city center (Ferraz) or to the Mostoles campus. A handful of learners with remarkable final marks in the high-school stage registered for the dual degree in the Mostoles campus. In Table 3, the frequency table is documented.

Table 3. Frequency table to second approach.

Campus	Frequency	Percent
Dual	28	22.6%
Ferraz	41	33.1%
Mostoles	55	44.4%
Total	124	100%

4.1. Phase One Results: All Students

In presenting the results of the research, we will first present results from the cohort in its entirety, the 124 students in total. Due to the number of samples in the dataset, N = 124, a Kolmogorov–Smirnov test was carried out assessing the distribution of both datasets using the total score accomplished, which was 0–16 points. As shown in Table 4, both test variables, pre and post, were normal distributions. All tests were conducted using an alpha level of 0.05. Consequently, parametric tests could be used to assess the difference between the scores achieved before and after using the Guided Scratch VEE.

Table 4. One-sample table: Kolmogorov–Smirnov test to pre- and post datasets.

	Pre-Test	Post-Test
N	124	124
Kolmogorov–Smirnov Z	0.937	0.794
p-value (95% significance)	0.344	0.553

A paired-sample *t*-test was used to contrast the score obtained before and after using the application. In Table 5, the relevant scores about the Student's *t*-test are shown. The outcomes indicated an extremely statistically significant difference before using the app, $\mu_{pre} = 2.98$ and $\sigma_{pre} = 1.73$, compared to after using the app, $\mu_{post} = 9.42$ and $\sigma_{post} = 2.48$, with Student's *t*-test = -30.303 and *p*-value = 0.1×10^{-13}.

Table 5. Paired samples statistics: outcomes of the Student's *t*-test for all students.

	Mean	N	Std. Deviation	Std. Error Mean
Pair 1, Pre-test	2.9778	124	1.72629	0.15503
Pair 1, Post-test	9.4238	124	2.47579	0.22233

	Paired Differences							
				95% Confidence Interval of the Difference				
	Mean	Std. Dev.	Std. Error Mean	Lower	Upper	t	df	Sig (2-tailed)
Pair 1 Pre-test, Post-test	−6.44	2.36	0.212	−6.86	−6.02	−30.3	123	0.0

Figure 4 illustrates the graphical comparison between scores achieved before and after, and Table 6 shows the descriptive statistics. It seems that there was a significant difference in respect of the two distributions. The median of the left boxplot, pre-test, was equal to three. Therefore, 50% of learners did not reach a score greater than three out of sixteen total scores. A group corresponding to the fourth quartile (75%) of all students achieved a score greater than four, but the maximum score obtained was 8.25 out of 16 points.

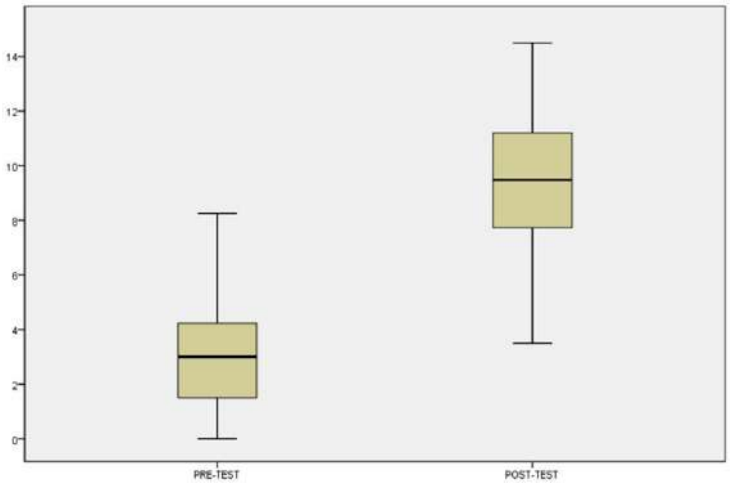

Figure 4. Pre-test and post-test of CS1 students after using Guided TPACK VEE.

On the other hand, the first quartile showed that a group of students reached a score of fewer than 2 points out of 16. Thus, the pre-test scores seem to indicate that students did not have enough programming knowledge. According to the second boxplot, post-test,

the outcomes achieved improved remarkably, with a median of 9.47 points. The variation between pre- and post-tests was 6.47 points. The first and last quartiles were bigger than the second and third. The maximum score attained was 14.50 out of 16 points, and the last quartile improved until 11.20 points. Therefore, 25% of students achieved a score equal to or greater than 11.20 points. On the contrary, the lowest score was 3.5 points out of 16 points, and the first quartile was 7.71.

Table 6. Descriptive statistics of results of students.

						Percentiles	
	N	**Mean**	**Std. Dev.**	**Min**	**Max**	**25th**	**50th**	**75th**
Pre-test	124	2.9778	1.726	0.0	8.25	1.5000	3.0000	4.2375
Post-test	124	9.4238	2.475	3.50	14.5	7.7125	9.4750	11.200

4.2. Phase Two Results: All Students and Their Understanding of Programming Concepts

Once a significant difference was observed between pre- and post-tests using the application in absolute terms, the next step was to determine which concepts demonstrated a change in learning, in other words, which had a remarkable improvement before and after using the Guide Scratch VEE. It was therefore necessary to review which concepts were more difficult to understand for the students.

To carry out this second part, the first thirteen answers of the rubric or test were considered. All questions were clustered into eight groups such as sequences, data flow, variables and types, operators, conditions, loops, events, and parallelism. The total score in this approach was 13 points. Additionally, the reliability of the items in the survey was very high. Cronbach's alpha was carried out, obtaining a value equal to 0.887, in other words, 89% of the variance in the scores was reliable, as shown in Table 7.

Table 7. Reliability statistics for the eight items in the survey.

Cronbach's Alpha	**Cronbach's Alpha Based on Standardized Items**	**N of Items**
0.887	0.900	8

According to the number of samples, N = 124, the one-sample Kolmogorov–Smirnov test was carried out for each programming concept and kind of test (pre and post). In all cases, *p*-values obtained were less than the level of significance (0.05). This seems to indicate that data were not from a normally distributed population. Thus, a non-parametric Wilcoxon signed-rank (pair sample) test was conducted to assess the improvement before and after using the app. The outcomes revealed that there was a statically significant variation between the score obtained before and after, as shown in Table 8. In all tests, the null hypothesis was rejected. Hence, this shows that distributions were different.

Table 8. Signed-rank test for the complete population and grouped by programming concepts.

Wilcoxon Signed-Rank Test	**W**	**Level of Significance**
Sequences	−5.996	0.000
Flow Chart	−9.240	0.000
Variables and Types	−9.451	0.000
Operations	−8.811	0.000
Conditionals	−7.836	0.000
Loops	−8.255	0.000
Events	−9.205	0.000
Parallelism	−9.072	0.000

In Figure 5, the graphical difference between pre- and post-distributions with respect to programming concepts is shown. The pre-test of the sequence concept indicated that

just 25% of students achieved the maximum score in this answer. However, in the post-test, more than 50% of learners obtained the maximum score (1 point). The following two concepts (flow charts variables and types of data) were an extreme case of study because, in the pre-test, more than 75% of learners did not reach a score greater than 0.75 out of 2 points. On the other hand, the post-test showed a remarkable improvement because more than 50% achieved at least 1.5 points out of 2 points. The operators' concept was the first programming concept with a notable number of outliers. Although the scores obtained in the post-test were better than the previous one, it had low scores. The fifth concept, conditionals, had a participant who achieved the maximum score in the pre-test. However, only 25% of students achieved a score greater than 0.75. The outcomes of the post-test showed an important advance in the scores accomplished. The following concept, loops, was another case of success of using the VEE because just 25% of learners did not obtain at least a score greater than one point. Finally, the last two programming concepts (events and parallelism), indicated that participants lacked pieces of knowledge about these topics. Only a handful of students obtained a score distinct to zero at the pre-test, but the improvement after the evaluation was also remarkable.

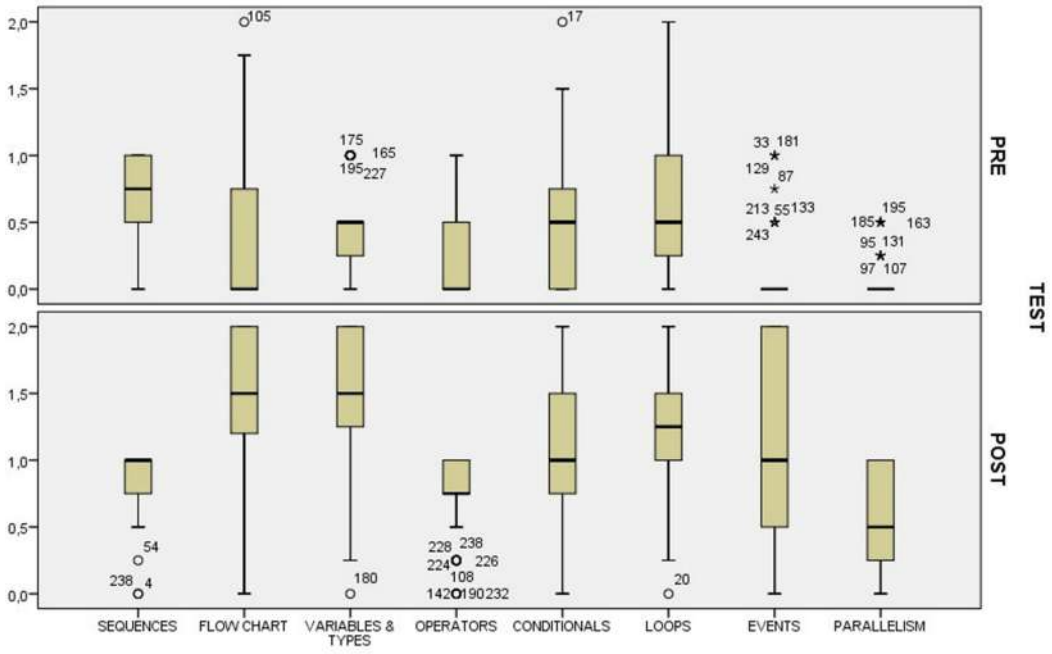

Figure 5. Boxplots of programming concepts from pre- and post-tests for all students. Notice that * and ○ are outliers in the distribution.

Therefore, before using the Guided Scratch VEE, students lacked knowledge about flow charts, variables, and data types, and after using the VEE, they greatly improved. In a similar vein, learning related to operators, conditional, and loops also improved, but their initial knowledge was also higher. However, in the case of events and parallelism, students did not have significant prior knowledge in the pre-test, and after using the VEE, they very much improved their knowledge.

4.3. Phase Three Results: Student Group and Understanding of Programming Concepts

Due to the fact that the dataset was divided into three groups (Ferraz, Mostoles, and Dual), it was necessary to overhaul the normal distributions of three sub-datasets. Ferraz and Dual had less than 50 samples in their respective corpus. Thus, the Shapiro–Wilk test of normality was carried out with the same *p*-value that in the previous section, in other words, a *p*-value less than 0.05. Moreover, the same situation was with the last group, Mostoles, but in this case, the number of samples was 55. Therefore, the one-sample Kolmogorov–Smirnov test was performed with the same output. It can also be noticed,

a non-parametric Wilcoxon signed-rank (pair sample) test was accomplished to evaluate the enhancement before and after employing the app for the three clusters. The results presented that there was a statically relevant variation between both distributions for all programming concepts, except to the first (sequences) of the Dual group, as shown in Table 9. The level of significance was greater than 0.05, but there was very little difference (p-value = 0.054). This fact could be produced because of the separate groups. The group of learners in the Dual group achieved better marks in their high-school stage than the others, and this may signify they had more prior knowledge in regard to programming at the beginning of the test. However, the value was not too high, and this kind of discrepancy only occurred once to assert something like this. The rank sum test results are depicted. The Wilcoxon test measures if medians of both distributions are equal or not. If they are different, the test rejects the null hypothesis; otherwise, the null hypothesis is accepted.

Table 9. Output of statistics of Wilcoxon signed-rank test for the sequence programming concept and Dual group.

	Sequences_Pre-Sequences_Post
Z	−1.931 [c]
Level of significance	0.054

Campus = dual degree in all campuses; Wilcoxon signed-rank test; Based on positive ranks.

A graphical representation of the results is shown in Figure 6. According to the Dual group, in the pre-test, programming knowledge was higher than in the Mostoles and Ferraz groups. The possible reasons were previously explained, or perhaps these participants studied more programming or technology subjects at high school, or they were simply better prepared. The Ferraz group achieved the lowest scores with respect to the three groups. Nevertheless, the post-test results demonstrated remarkable development out of all groups. For the Dual group, compared to the other groups, their pre-test was higher, as well as their post-test in all the programming concepts. It is also noteworthy that the trajectory of the learning in all three groups follows the same path, even though results are higher, respectively, in Dual, Mostoles, and Ferraz. Corresponding to the grade, each group follows the same order (higher for Dual, Mostoles, and Ferraz, respectively).

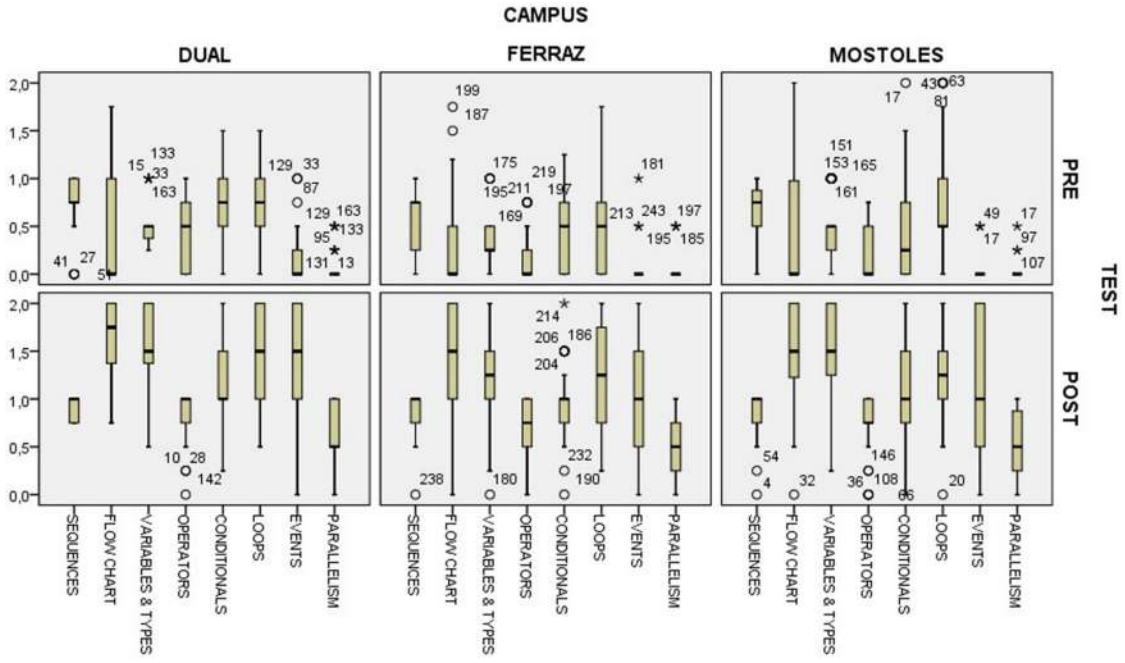

Figure 6. Boxplots of programming concepts from pre- and post-tests divided into three groups. Notice that * and ○ are outliers in the distribution.

As this study follows the same trajectory for the learning of programming concepts, the results shown in the previous section are equivalent. The programming concepts with the best performance in relation to the pre- and post-test are the event and parallelism concepts. The concept with the least knowledge gained is sequences, as the participants were familiar with it, followed by flowcharts, variables, and types, and then operators and conditionals. In most cases, the pre-test presented scores close to zero (except for some outliers in all groups).

4.4. Discussion of Findings

Results obtained from this research support previous studies [24], where participants gained relevant knowledge on also CT using games. This also correlates with other work, where an environment for learning programming such as Scratch was used to ensure motivation and empowerment of students [4], and these positive effects of the Guided Scratch VEE match earlier research studies [32–34]. Previous approaches [40] have also used practical games [41] or math-proposed games for improving CT, as in this study.

Therefore, providing visual environments based on interactive games for introducing such programming concepts and CT concepts to students may be the correct direction to work with, or at least a very good recommendation.

Nonetheless, a limitation of this research is based on the lack of a control group to check or extrapolate the outcomes accomplished. This fact does not have a trivial solution because the number of students and groups is limited. A plausible solution was the use of the smallest campus (e.g., Ferraz), as the control group, but the authors considered a better option, consisting of an increase in samples instead of a control group.

5. Conclusions

The current COVID-19 pandemic has served to underscore the importance of enhancing young people's ability, understanding, and use of computer code to develop and deploy new software systems. Productive computer programmers must be able to apply general practices and concepts involved in computational thinking and problem solving. That said, computer science education represents a dynamically changing domain globally and has faced many challenges in teaching programming concepts. This paper uses a guided TPACK framework, incorporating a Scratch VEE for CS1 students as a method to teach and introduce programming concepts. The objective was to investigate if programming concepts could be improved upon by applying this approach and if the CS1 students studying at the one university but different groups of students of the same undergraduate degree were able to improve their programming skills.

This paper had two research questions, the first being, 'Can programming concepts be improved with a TPACK Visual Execution Environment for a cohort of 124 of CS1 students?' Statistically significant results of their knowledge improvement in both cases were observed, studying the results as a whole (all students) or dividing them into their own groups, Ferraz, Mostoles, and both, in which case the results vary slightly depending on their previous knowledge and precedence. The second research question focused on if there were differences in the learning path of programming concepts, which could infer that some concepts are easier to understand by students than others. The study observed that there are differences depending on the concept being addressed and their initial knowledge of each. It was observed that students lacked previous knowledge about flow charts, variables, and data types, which they showed great improvement. Slightly the same thing happened with operators, conditionals, and loops, but their initial knowledge was also higher. Moreover, in the case of events and parallelism, students did not know about them at all in the pre-test, and after using the VEE, their knowledge very much improved.

Another conclusion observed after the study is that 'origin' matters to CS1 students. The three groups of students followed the same procedure and had the same teachers, but their initial knowledge and knowledge gain were not the same for the three groups. In

evaluating their prior knowledge from high-school grades on accessing university, this indicated more prior knowledge for the groups that scored higher.

The work, being a quasi-experimental research case study with CS1 students, though with different groups of students belonging to the same degree, was conducted in one country and therefore has the primary limitation of a narrow focus. This can effectively point to results requiring further evaluation; however, such an approach does not facilitate the development of generalizations.

While this VEE is not a panacea to all programming comprehension and learning, this paper has demonstrated that by using metaphors and Scratch programs to explain and practice the programming concepts addressed in this paper, the VEE effectively guides CS1students in learning programming concepts in a short period of time. While ensuring concepts and practices foundational to programming are understood, the students can continue their practice and become competent and productive computer programmers.

Author Contributions: R.H.-N.; investigation, R.H.-N., D.P.-A., C.C. and O.B.-G.; resources, R.H.-N., C.C., D.P.-A. and O.B.-G.; data curation, R.H.-N., C.C. and O.B.-G.; writing—original draft–preparation, R.H.-N., D.P.-A. and O.B.-G.; writing—review and editing, R.H.-N. and C.C.; visualization, R.H.-N. and D.P.-A.; supervision, R.H.-N. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by in part by the Ministerio de Economía y Competitividad under Grant TIN2015-66731-C2-1-R, in part by the Rey Juan Carlos University under Grant 30VCPIGI15, in part by the Madrid Regional Government, through the project e-Madrid-CM, under Grant P2018/TCS-4307, and in part by the Structural Funds (FSE and FEDER).

Data Availability Statement: All datasets are available and can be requested from the authors.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lau, W.W.; Yuen, A.H. Modelling programming performance: Beyond the influence of learner characteristics. *Comput. Educ.* **2011**, *57*, 1202–1213. [CrossRef]
2. Campe, S.; Denner, J. *Programming Games for Learning: A Research Synthesis*; American Educational Research Association (AERA): Chicago, IL, USA, 2015.
3. Jovanov, M.; Stankov, E.; Mihova, M.; Ristov, S.; Gusev, M. Computing as a new compulsory subject in the Macedonian primary schools curriculum. In Proceedings of the 2016 IEEE Global Engineering Education Conference (EDUCON), Abu Dhabi, United Arab Emirates, 10–13 April 2016.
4. Ouahbi, I.; Kaddari, F.; Darhmaoui, H.; Elachqar, A.; Lahmine, S. Learning Basic Programming Concepts by Creating Games with Scratch Programming Environment. *Procedia Soc. Behav. Sci.* **2015**, *191*, 1479–1482. [CrossRef]
5. Lahtinen, E.; Ala-Mutka, K.; Järvinen, H.-M. A study of the difficulties of novice programmers. *ACM SIGCSE Bull.* **2005**, *37*, 14–18. [CrossRef]
6. Ginat, D. On Novice Loop Boundaries and Range Conceptions. *Comput. Sci. Educ.* **2004**, *14*, 165–181. [CrossRef]
7. Seppälä, O.; Malmi, L.; Korhonen, A. Observations on student misconceptions—A case study of the Build—Heap Algorithm. *Comput. Sci. Educ.* **2006**, *16*, 241–255. [CrossRef]
8. Barker, L.J.; McDowell, C.; Kalahar, K. Exploring factors that influence computer science introductory course students to persist in the major. *ACM SIGCSE Bull.* **2009**, *41*, 153–157. [CrossRef]
9. Coull, N.J.; Duncan, I. Emergent Requirements for Supporting Introductory Programming. *Innov. Teach. Learn. Inf. Comput. Sci.* **2011**, *10*, 78–85. [CrossRef]
10. Yadav, A.; Gretter, S.; Hambrusch, S.; Sands, P. Expanding computer science education in schools: Understanding teacher experiences and challenges. *Comput. Sci. Educ.* **2016**, *26*, 235–254. [CrossRef]
11. Yadav, A.; Mayfield, C.; Zhou, N.; Hambrusch, S.; Korb, T. Computational Thinking in Elementary and Secondary Teacher Education. *ACM Trans. Comput. Educ.* **2014**, *14*, 1–16. [CrossRef]
12. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*; Basic Books: New York, NY, USA, 1980.
13. Astin, W.A. *College Retention Rates Are often Misleading*; Chronicle of Higher Education: Washington, DC, USA, 1993.
14. Stuart, V.B. Math Course or Math Anxiety? *Natl. Counc. Teach. Math.* **2000**, *6*, 330.
15. Piaget, J. *The Moral Judgement of the Child*; Penguin Books: New York, NY, USA, 1932.
16. Piaget, J. *Origins of Intelligence in Children*; International Universities Press: New York, NY, USA, 1952.
17. Vygotsky, L.S. *Thought and Language*, 2nd ed.; MIT Press: Cambridge, MA, USA, 1962.
18. Vygotsky, L.S. *Mind in Society: The Development of Higher Psychological Process*; Harvard University Press: Cambridge, MA, USA, 1978.

19. Vygotsky, L.S. The Genesis of Higher Mental Functions. In *Cognitive Development to Adolescence*; Richardson, K., Sheldon, S., Eds.; Erlbaum: Hove, UK, 1988.
20. Maleko, M.; Hamilton, M.; D'Souza, D. Novices' Perceptions and Experiences of a Mobile Social Learning Environment for Learning of Programming. In Proceedings of the 12th International Conference on Innovation and Technology in Computer Science Education (ITiCSE), Haifa, Israel, 3–5 July 2012.
21. Williams, L.; Wiebe, E.; Yang, K.; Ferzli, M.; Miller, C. In Support of Pair Programming in the Introductory Computer Science Course. *Comput. Sci. Educ.* **2002**, *12*, 197–212. [CrossRef]
22. Renumol, V.; Jayaprakash, S.; Janakiram, D. *Classification of Cognitive Difficulties of Students to Learn Computer Programming*; Indian Institute of Technology: New Delhi, India, 2009; p. 12.
23. De Jong, I.; Jeuring, J. Computational Thinking Interventions in Higher Education. In Proceedings of the 20th Koli Calling International Conference on Computing Education Research, Koli, Finland, 19–22 November 2020.
24. Agbo, F.J.; Oyelere, S.S.; Suhonen, J.; Laine, T.H. Co-design of mini games for learning computational thinking in an online environment. *Educ. Inf. Technol.* **2021**, *26*, 5815–5849. [CrossRef] [PubMed]
25. Jenkins, T. The motivation of students of programming. *ACM SIGCSE Bull.* **2001**, *33*, 53–56. [CrossRef]
26. Kurland, D.M.; Pea, R.D.; Lement, C.C.; Mawby, R. A Study of the Development of Programming Ability and Thinking Skills in High School Students. *J. Educ. Comput. Res.* **1986**, *2*, 429–458. [CrossRef]
27. Brooks, F.P. No Silver Bullet: Essence and Accidents of Software Engineering. In Proceedings of the Tenth World Computing Conference, Dublin, Ireland, 1–5 September 1986; pp. 1069–1076.
28. Mishra, D.; Ostrovska, S.; Hacaloglu, T. Exploring and expanding students' success in software testing. *Inf. Technol. People* **2017**, *30*, 927–945. [CrossRef]
29. Clancy, M.J.; Linn, M.C. Case studies in the classroom. *ACM SIGCSE Bull.* **1992**, *24*, 220–224. [CrossRef]
30. Chandramouli, M.; Zahraee, M.; Winer, C. A fun-learning approach to programming: An adaptive Virtual Reality (VR) platform to teach programming to engineering students. In Proceedings of the IEEE International Conference on Electro/Information Technology, Milwaukee, WI, USA, 5–7 July 2014.
31. Silapachote, P.; Srisuphab, A. Teaching and learning computational thinking through solving problems in Artificial Intelligence: On designing introductory engineering and computing courses. In Proceedings of the 2016 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), Bangkok, Thailand, 7–9 December 2016.
32. Liu, C.-C.; Cheng, Y.-B.; Huang, C.-W. The effect of simulation games on the learning of computational problem solving. *Comput. Educ.* **2011**, *57*, 1907–1918. [CrossRef]
33. Kazimoglu, C.; Kiernan, M.; Bacon, L.; Mackinnon, L. A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. *Procedia Soc. Behav. Sci.* **2012**, *47*, 1991–1999. [CrossRef]
34. Kazimoglu, C.; Kiernan, M.; Bacon, L.; MacKinnon, L. Learning Programming at the Computational Thinking Level via Digital Game-Play. *Procedia Comput. Sci.* **2012**, *9*, 522–531. [CrossRef]
35. Saad, A.; Shuff, T.; Loewen, G.; Burton, K. Supporting undergraduate computer science education using educational robots. In Proceedings of the ACMSE 2018 Conference, Tuscaloosa, AL, USA, 29–31 March 2012.
36. Weintrop, W.; Wilensky, U. Comparing Block-Basedand Text-Based Programming in High School Computer Science Classrooms. *ACM Trans. Comput. Educ.* **2017**, *18*, 1. [CrossRef]
37. Martínez-Valdés, J.A.; Velázquez-Iturbide, J.; Neira, R.H. A (Relatively) Unsatisfactory Experience of Use of Scratch in CS1. In Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality, Cadiz, Spain, 18–20 October 2017.
38. Aristawati, F.A.; Budiyanto, C.; Yuana, R.A. Adopting Educational Robotics to Enhance Undergraduate Students' Self-Efficacy Levels of Computational Thinking. *J. Turk. Sci. Educ.* **2018**, *15*, 42–50.
39. Basu, S.; Biswas, G.; Kinnebrew, J.S. Learner modeling for adaptive scaffolding in a Computational Thinking-based science learning environment. *User Model. User-Adapted Interact.* **2017**, *27*, 5–53. [CrossRef]
40. Benakli, N.; Kostadinov, B.; Satyanarayana, A.; Singh, S. Introducing computational thinking through hands-on projects using R with applications to calculus, probability and data analysis. *Int. J. Math. Educ. Sci. Technol.* **2016**, *48*, 393–427. [CrossRef]
41. Cai, J.; Yang, H.H.; Gong, D.; MacLeod, J.; Jin, Y. A Case Study to Promote Computational Thinking: The Lab Rotation Approach. In *Blended Learning: Enhancing Learning Success*; Cheung, S.K.S., Kwok, L., Kubota, K., Lee, L.K., Tokito, J., Eds.; Springer: Cham, Switzerland, 2018; pp. 393–403.
42. Dodero, J.M.; Mota, J.M.; Ruiz-Rube, I. Bringing computational thinking to teachers' training. In Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality, Cádiz, Spain, 18–20 October 2017.
43. Gabriele, L.; Bertacchini, F.; Tavernise, A.; Vaca-Cárdenas, L.; Pantano, P.; Bilotta, E. Lesson Planning by Computational Thinking Skills in Italian Pre-service Teachers. *Inform. Educ.* **2019**, *18*, 69–104. [CrossRef]
44. Curzon, P.; McOwan, P.W.; Plant, N.; Meagher, L.R. Introducing teachers to computational thinking using unplugged storytelling. In Proceedings of the 9th Workshop in Primary and Secondary Computing Education, Berlin, Germany, 5 November 2014; pp. 89–92.
45. Jaipal-Jamani, K.; Angeli, C. Effect of robotics on elementary preservice teachers' self-efficacy, science learning, and computational thinking. *J. Sci. Educ. Technol.* **2017**, *26*, 175–192. [CrossRef]

46. Hsu, T.-C.; Chang, S.-C.; Hung, Y.-T. How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Comput. Educ.* **2018**, *126*, 296–310. [CrossRef]
47. Fogg, B.J. A behavior model for persuasive design. In Proceedings of the 4th international Conference on Persuasive Technology, Claremont, CA, USA, 26–29 April 2009; pp. 1–7.
48. Piaget, J.; Inhelder, B. *Memory and Intelligence*; Basic Books: New York, NY, USA, 1973.
49. Mishra, P.; Koehler, M. Technological Pedagogical Content Knowledge: A Framework for Teacher Knowledge. *Teach. Coll. Rec.* **2006**, *108*, 1017–1054. [CrossRef]
50. Brennan, K.; Resnick, M. *New Frameworks for Studying and Assessing the Development of Computational Thinking*; American Educational Research Association: Vancouver, BC, Canada, 2012.
51. Mishra, P.; Koehler, M.J. *Introducing Technological Pedagogical Content Knowledge*; American Educational Research Association: Vancouver, BC, Canada, 2008; pp. 1–16.
52. Diéguez, J.L.R. *Metaphors in Teaching*; Revista Interuniversitaria de Didáctica, Universidad de Salamanca: Salamanca, Spain, 1988; pp. 223–240.
53. Bouton, C.L. Nim, a Game with a Complete Mathematical Theory. *Ann. Math.* **1901**, *3*, 35. [CrossRef]
54. Ramírez, S.U. Informática y teorías del aprendizaje. Píxel-Bit. *Rev. Medios Educ.* **1999**, *12*, 87–100.